TP 6: Objets

Objectifs:

- Savoir développer un répertoire téléphonique
- Utilisation d'objets

Objets JavaScript

Dans la première moitié de ce chapitre nous avons vu que les tableaux permettaient de stocker une liste ordonnée d'éléments (ayant chacun une valeur de n'importe quel type). Et que ces éléments étaient indexés par des numéros.

Exemple:

```
var tableau = ['a', 'b', 4];
// => Correspondance entre numéros d'éléments et leur valeur
//    0: 'a'
//    1: 'b'
//    2: 4
```

Les objets permettent aussi de stocker des valeurs.

Alors que chaque valeur d'un tableau est stockée dans un **élément**, chaque valeur d'un objet est stockée dans ce qu'on appelle une **propriété**.

Alors que chaque élément d'un tableau est indexé par un numéro (indice), chaque propriété d'un objet est indexé par une chaîne de caractères.

Définition d'objet

Contrairement à la manière de définir un tableau, chaque valeur de propriété stockée dans un objet doit être précédée du nom de cette propriété (aussi appelé **clé**, ou **key** en anglais), et suivi par : (deux points).

```
var objet = {
  prop1: 'a',
  prop2: 'b',
  prop3: 4,
};
// => Correspondance entre noms de propriétés et leur valeur
//    'prop1': 'a'
//    'prop2': 'b'
//    'prop3': 4
```

À noter que chaque définition de propriété doit être ponctuée d'une virgule.

Les règles d'indentation s'appliquent aussi à la définition d'objets sur plusieurs lignes, car les propriétés sont définies entre accolades.

Le format JSON (JavaScript Object Notation, utilisé très couramment par les APIs de sites web, et exports structurées de données depuis le web) correspond à cette manière de définir des objets. Nous allons y revenir plus tard dans ce chapitre.

Accès aux propriétés d'un objet

Il existe deux manière d'accéder à la valeur d'une propriété d'objet:

1. La notation pointée, ex: `objet.prop`
2. L'usage des crochets, ex: `objet[prop]`

Comme pour les tableaux, l'adressage par crochets a un avantage intéressant: il permet d'accéder à un élément dont la clé est stockée dans une variable.

Exemple:

```
var objet = {
  prop1: 'a',
  prop2: 'b',
  prop3: 4,
};

// adressage littéral par notation pointée:
objet.prop2;        // => 'b'

// adressage littéral par crochets:
objet['prop2'];     // => 'b'

// adressage symbolique par crochets:
var cle = 'prop2';
objet[cle];         // => 'b'
```

La notation pointée est plus concise et lisible, mais impose de mentionner la clé "en dur" (littéralement) dans le code:

```
// adressage littéral, avec notation pointée:
objet.prop2; // => 'b'

// adressage symbolique, impossible avec notation pointée:
var cle = 'prop2';
objet.cle;    // => undefined, car il n'y a pas de propriété appelée 'cle' dans l'objet
```

Modification de propriétés

La modification d'une propriété d'objet fonctionne similairement à la modification d'un élément de tableau: il suffit d'effectuer une affectation après avoir adressé la valeur à modifier.

```
var tableau = ['a', 'b', 4];
tableau[1] = 3; // => tableau = ['a', 3, 4]

var objet = {
  prop1: 'a',
  prop2: 'b',
  prop3: 4,
};
objet.prop2 = 3; // => objet: { prop1: 'a', prop2: 3, prop3: 4 }
```

Il est bien sûr possible d'utiliser indifféremment l'adressage par notation pointée ou par crochets, pour une affectation.

Clés riches

Contrairement aux restrictions imposées pour nommer les variables en JavaScript, il est possible d'inclure des espaces et des caractères spéciaux dans les noms de propriétés.

Par contre, il faut dans ce cas utiliser des apostrophes (ou guillemets) pour définir et adresser ces propriétés:

```
var mesAmis = {
  'Luke Skywalker': true,
  'Dark Vador': false,
};
mesAmis.LukeSkywalker;     // => undefined
mesAmis.Luke Skywalker;    // => /!\ syntax error
mesAmis['Luke Skywalker']; // => true
```

Notes sur les types

Comme pour les tableaux, la valeur d'une propriété d'objet peut être de n'importe quel type: `string`, `number`, `boolean`, `null`, `undefined`, `object` ou `function`.
A noter que, comme les objets, les tableaux sont aussi de type `object`.

```
typeof { prop: 'a' };  // => "object"
typeof [ 'a', 3, 4 ];  // => "object"
```

Mais les tableaux ont des caractéristiques particulières que n'ont pas les objets. Notamment l'indexation numérique, des fonctions spécifiques comme `slice`, etc...

Mise en pratique: Annuaire téléphonique

But: développer un programme qui permet d'afficher le numéro de téléphone d'un ami (à l'aide de `alert`) dont le nom a été saisi par l'utilisateur (à l'aide de `prompt`).

Afin que l'annuaire soit extensible à l'avenir, aucune condition `if` ne doit être utilisée. Nous allons donc stocker les noms et numéros de téléphones dans un objet, et utiliser l'adressage par crochets pour trouver un numéro à partir d'un nom.

```
var annuaire = {
  patricia: '06 66 66 66 66',
  david: '07 77 77 77 77',
};
```

1. Afficher dans la console le numéro de téléphone de `patricia`, en utilisant la notation pointée sur l'objet `annuaire`;
2. Demander à l'utilisateur de saisir un prénom, puis afficher le numéro de téléphone associé à ce prénom.

---

4. Objets, usage avancé

Hiérarchie d'objets

Comme il est possible de stocker un objet comme valeur d'une propriété d'objet, cela veut dire que nous pouvons définir des hiérarchies / arbres d'objets.

Exemple:

```
var compteFacebook = {
  amis: {
    'Luke Skywalker': true,
    'Dark Vador': false,
  },
  groupes: {
    maitresJedi: {
      titre: 'Groupe des maîtres Jedi',
      membres: [ 'Yoda', 'Obi Wan' ],
    },
    lolcats: {
      titre: 'Vive les chats !',
      membres: [ 'Patrick' ],
    },
  },
};
```

Dans l'exemple ci-dessus, nous avons jusqu'à trois niveaux d'imbrication d'objets (voire quatre, si on compte les tableaux comme des objets): l'objet `compteFacebook` contient une propriété `groupes` (de type objet) contenant une propriété `maitresJedi` (aussi de type objet) contenant deux propriétés (de type chaîne de caractères et tableau).
Cet objet peut être représenté visuellement sous forme d'un arbre.

Pour accéder au premier élément du tableau `membres` de la sous-propriété `maitresJedi`, on doit écrire le cheminement à suivre, de propriété à sous-propriétés.

Exemple:

```
// adressage par notation pointée:
compteFacebook.groupes.maitresJedi.membres[0]; // => 'Yoda'

// adressage par crochets:
compteFacebook['groupes']['maitresJedi']['membres'][0]; // (idem)
```

Dans cet exemple aussi, on peut utiliser indifféremment la notation pointée ou les crochets, pour accéder à la propriété d'un objet.

Énumérer les propriétés d'un objet

Il existe deux manières d'énumérer les propriétés d'un objet:

1. utiliser une boucle `for-in`;
2. ou itérer sur le tableau des clés de l'objet, à l'aide de la fonction `Object.keys()`.

Une boucle `for-in` s'utilise de cette façon:

```
var mesAmis = {
  'Luke Skywalker': true,
  'Dark Vador': false,
};
for (var cle in mesAmis) {
  console.log(cle, '->', mesAmis[cle]);
}
// => lignes affichées dans la console:
//    Luke Skywalker -> true
//    Dark Vador -> false
```

Contrairement à une boucle `for` classique (*dans laquelle on définit une instruction d'initialisation, une expression conditionnelle et une instruction d'incrémentation, séparés par des points-virgules, pour rappel*), la boucle `for-in` va répéter la liste d'instructions entre accolades pour chaque clé de l'objet.
Dans notre exemple de code ci-dessus, la variable `cle` prendra donc la valeur d'une clé de l'objet `mesAmis`, pour chacune de ses propriétés.
A noter que cela fonctionne aussi sur les tableaux, sauf que l'indice de chaque élément du tableau sera fourni (au lieu de la clé de chaque propriété de l'objet).

Énumérer les propriétés d'un objet, une autre manière

Lorsqu'elle est appelée en passant un objet en paramètre, la fonction `Object.keys()` retourne un tableau contenant les clés/noms de chaque propriété de cet objet.

```
var mesAmis = {
  'Luke Skywalker': true,
  'Dark Vador': false,
};
var cles = Object.keys(mesAmis); // => [ 'Luke Skywalker', 'Dark Vador' ]
```

On peut alors utiliser une boucle `for` plus classique pour itérer sur les valeurs de ce tableau:

```
for (var i = 0; i < cles.length; i++) {
  var cle = cles[i];
  console.log(i, ':', cle, '->', mesAmis[cle]);
}
// => lignes affichées dans la console:
//    0, Luke Skywalker -> true
//    1, Dark Vador -> false
```

Suppression d'une propriété

Pour supprimer une propriété d'un objet, il faut utiliser le mot clé `delete` de la manière suivante:

```
var objet = {
  prop1: 'a',
  prop2: 'b',
  prop3: 4,
};
delete objet.prop2;
// => objet === { prop1: 'a', prop3: 4 }
```

Exercice: Répertoire téléphonique

Sur la base de l'annuaire téléphonique réalisé plus haut, développer un programme qui propose les fonctionnalités suivantes:

- recherche et affichage d'un numéro de téléphone en saisissant un nom,
- liste des contacts de l'annuaire (nom + numéro de téléphone, à afficher dans la console à l'aide d'une boucle `for..in`),
- ajout d'un contact (nom + numéro de téléphone) dans l'annuaire,
- suppression d'un contact.

Pour permettre à l'utilisateur de choisir la fonctionnalité qu'il souhaite utiliser, lui demander de saisir la première lettre de la fonction:

- `'r'` pour rechercher,
- `'l'` pour afficher la liste,
- `'a'` pour ajouter,
- et `'s'` pour supprimer.

Afin que l'utilisateur puisse utiliser plusieurs fonctionnalités d'affilée, placer le code du programme dans une boucle qui ne s'arrêtera que lorsque l'utilisateur aura tapé `q` au lieu de choisir une fonctionnalité.